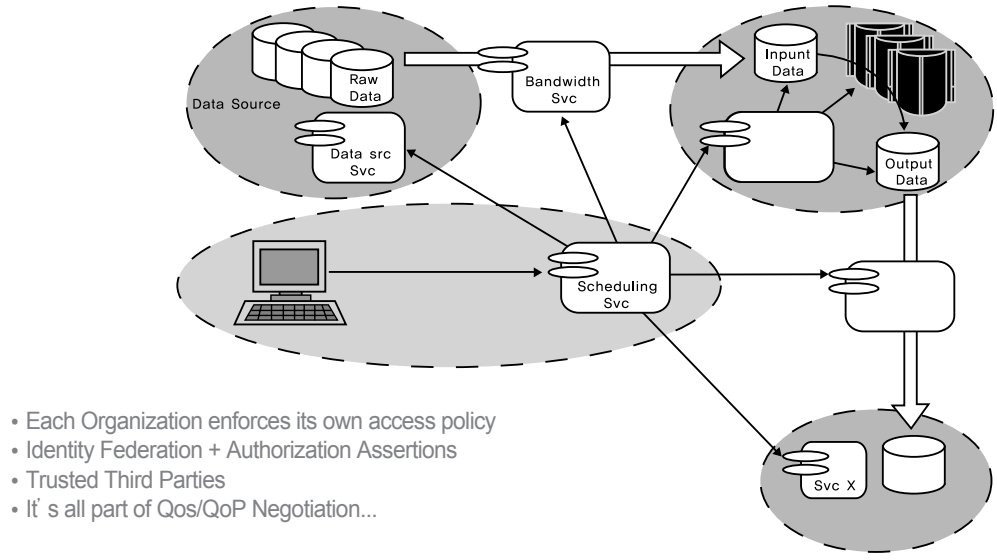
데이터 그리드 전송 핵심 – GridFTP

글 _ 허의남 교수 · 서울여자대학교 정보통신공학부 · huh@swu.ac.kr

1. 서론

고에너지 물리나 유전자 정보처리 분야 등의 많은 과학 분야에서 거대한 양의 데이터가 이용되고 있으며, 또한 많은 양의 데이터가 생산되고 있다. 이러한 대량의 데이터를 하나의 스토리지에 저장하기는 불가능하므로 분산된 여러 스토리지를 이용한다. 데이터 그리드는 이렇게 지역적으로 분산된 데이터를 통합, 관리, 분석할 수 있는 환경을 제공한다. 기존의 분산된 스토리지 관리 시스템은 서로 다른 프로토콜을 이용하므로 서로 다른 시스템 간에 호환성이 적다는 문제점을 안고 있었다. 그러나 데이터 그리드는 전 세계 통합 데이터 관리를 목표로 일반 인터넷에서 사용하는 파일 전송 프로토콜인 FTP를 이용함으로써 호환성의 문제를 해결하였다. 이러한 데이터 그리드에서는 대용량 데이터 전송을 위하여 고성능의 안전하고 견고한 데이터 전송 메커니즘, GridFTP를 핵심 기술로 포함하고 있다.



〈그림 1〉 그리드 서비스 연관성

위의 〈그림 1〉처럼 그리드에서는 대용량의 데이터 이동이 가상 도메인 간에 빈번히 일어나기에 밴드위스 서비스의 중요성은 앞으로 그 기능이 복잡해 질것이다. 예를 들어 SLA(Service Level Agreement)에 대응하는 대역폭을 가상기관 사이에서 유지해야 한다는 것이다.

GridFTP는 병렬 데이터 전송과 스트라이핑 데이터 전송 기능을 포함하고 있어, TCP로 인해 발생하는 전송 지연 문제를 해결하고 있다. 그러나 데이터 병렬 전송 시 생성되는 스트림 간에는 서로 간섭이 발생되며 이로 인해 전송 지연이 발생된다. 따라서 본 원고에서는 병렬전송의 효과를 실험을 통하여 보이고, 병렬 전송의 문제점과 해결책을 제기하고자 한다.

본 글은 다음과 같이 구성되어 있다. 2장에서는 GridFTP의 특징을 간략히 소개하고, 3장에서는 실험을 통하여 병렬 전송의 효과를 보이고 성능을 분석한다. 4장에서는 병렬 전송의 문제점을 제기하고, 5장에서는 결론과 향후 연구 과제를 제시한다.

2. GridFTP

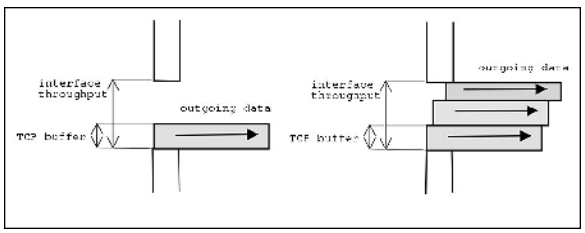
가. GridFTP 특징

(1) Parallel transfers

데이터 전송을 위한 TCP socket은 TCP window에 의해서 제한을 받으며, TCP buffer size에도 영향을 받는다. GridFTP는 TCP 기반으로 설계되어 있으므로 위의 TCP window나 TCP buffer size에 영향을 받게 된다. 일반적으로 TCP buffer size는 64KB로 설정되어 있으며 이는 네트워크 인터페이스 처리량을 최대로 조절하기에는 너무나 작은 크기이다. 따라서 TCP buffer size를 조절해야 네트워크 대역폭을 효율적으로 사용할 수 있다. 그러나 GridFTP는 TCP buffer size를 조절하지 않고 네트워크 자원을 최대한 사용하기 위해 병렬 전송 기술을 사용한다.

GridFTP는 동시 다중 연결을 지원함으로써 하나의 파일을 다중 스트림으로 전송할 수 있으며, 여기서 각각의 스트림을 data pathway라 부른다. 〈그림 2〉는 병렬 스트림을 이용하여 데이터를 전송하는 상태와 단일 스트림을 이용하여 데이터를 전송하는 상태를 비교하여 보여주고 있다.

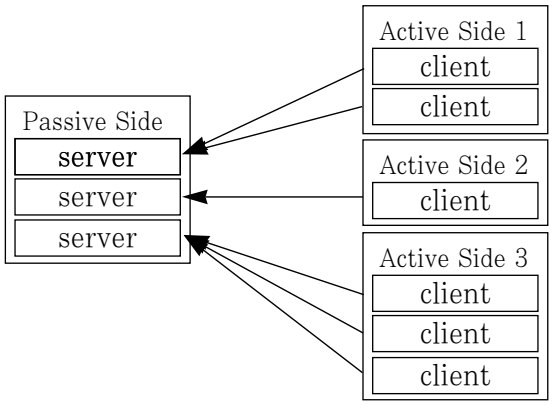
〈그림 2〉의 왼쪽은 단일 스트림을 이용하여 데이터를 전송하는 상태를 나타낸 것으로 네트워크 대역폭을 충분히 활용하지 못하고 있다. 그러나 오른쪽의 그림을 보면 TCP buffer 크기를 조절하지 않았음에도 불구하고 병렬 스트림에 의해 네트워크 대역폭이 모두 사용되는 상태를 볼 수 있다.



〈그림 2〉 단일 스트림을 이용한 데이터 전송과 병렬 스트림을 이용한 데이터 전송 비교(default TCP buffer size)
그림 출처 : http://www.icslab.agh.edu.pl/~kzajac/pawel/GridFTPexplained.doc

(2) Striped transfer

GridFTP는 스트라이핑(striping)을 지원한다. Striping은 여러 호스트 사이에 병렬로 데이터를 전송하기 위한 기술로, 병렬화(parallelism)와 striping 간의 중요한 차이점은 passive side(연결요청을 기다리는 호스트)에서 server들이 연결 요청을 기다릴 때, 하나의 주소 대신에 주소 배열을 가지고 있다는 것이다. 이들 각각의 passive server들은 연결 요청이 들어오기를 기다리며, 각 passive server들에게 연결 요청이 들어와 생성된 각 스트림을 strip이라 부른다. 이 기술은 하나의 데이터가 여러 호스트에 분산되어 저장되어 있는 경우 각 호스트에 스트림을 생성하여 쪼개진 데이터를 전송받을 수 있도록 지원해 준다.



〈그림 3〉 parallelism과 striping의 통합

또한, Parallelism과 striping은 통합될 수 있다. 〈그림 3〉은 통합된 parallelism과 striping을 나타내고 있다. 이런 경우 stripe은 passive server들 중에 하나와 형성된 연결들의 집합을 의미한다. 한 stripe은 여러 pathway들로 구성될 수 있고, 각 stripe은 parallelism으로 이루어진다.
더욱이 parallelism의 경우에는 성능 향상에 네트워크 인터페이스 하드웨어의 제한을 받으나 striping 같은 경우에는 이러한 제한을 넘어서 성능을 향상시키는 것이 가능하다.

나. GridFTP 소스 분석

GridFTP를 실행하여 원격 파일을 다운 받기 위해서는 GridFTPClient Class의 get() 메소드를 호출하도록 되어있다. GridFTPClient Class는 FTPClient Class로부터 상속을 받았으며, GridFTPClient Class의 get() 메소드는 상위 클래스의 get() 메소드를 호출하도록 구현되어 있다. get() 메소드 내부를 보면, setPassive()와 setLocalActive() 메소드를 이용하여 passive side와 active side를 설정하고, 그 후에 actualGet() 메소드를 호출하고 있다. actualGet() 메소드는 store() 메소드를 호출하도록 구현되어 있으며, store() 메소드는 GridFTPServerFacade Class에 정의되어 있다. 아래에 store() 메소드 구현 부분이 나와 있다.

```
#################################################################
GridFTPSeverFacade Class
#################################################################
public class GridFTPServerFacade extends FTPServerFacade{
.....
public void store(DataSink sink) {
.............................
              for (int i = 0; i < gSession.parallel; i++) {
                    logger.debug("creating data connection task");
             runTask(createPassiveConnectTask(sink, context));
              }
.........................
}
```

gSession.parallel에 저장된 정수만큼 반복하면서 runTask() 메소드를 호출하고 있다. gSession은 두 호스트 간에 연결될 때 필요한 정보들을 저장하고 있는 인스턴스로, 사용자가 입력한 병렬 스트림의 개수가 이 인스턴스의 parallel 변수에 저장된다. 위의 소스를 보면, gSession.parallel 동안 반복하면서 task(데이터 전송 작업)를 생성하도록 구현되어 있다. runTask() 메소드는 다음과 같이 정의되어 있다.

```
#################################################################
FTPServerFacade Class
#################################################################
public class FTPServerFacade {
....
protected void runTask(Task task) {
          if (taskThread == null) {
              taskThread = new TaskThread();
          }
          taskThread.runTask(task);
    }
....
```

runTask() 메소드에서는 인자로 들어온 task 인스턴스를 버퍼에 저장하고 이를 하나씩 꺼내 수행시키는 역할을 하고 있다.

TaskThread()는 Runnable로부터 implements된 메소드로 thread이며, start(), run(), getNextTask() 등의 메소드를 이용하여 버퍼에 task를 저장하거나 저장되어 있던 task를 꺼내어 수행한다.
앞에서 설명한 GridFTPServerFacade Class에 정의된 runTask() 메소드를 보면 인자로 createPassiveConnectTask(sink, context)가 호출되고 있다. 이는 passiveConnectTask Class 로부터 상속된 것으로 실제 두 호스트 간에 연결을 형성하고 데이터를 전송하는 일을 수행한다. 다음은 PassiveConnetTask Class를 보여주고 있다.

```
##################################################
PassiveConnectTask Class
##################################################
public class PassiveConnectTask extends Task {
....
public void execute() {
          try {
        ① DataChannel dataChannel = null;
            ..................................................
                    mySocketBox = new SocketBox();
              ② mySocketBox.setSocket(openSocket());
.....................................................
                    dataChannel =  factory.getDataChannel(session, mySocketBox);
..............................................
               ③ dataChannel.startTransfer(sink, control, context);
.............................
   protected Socket openSocket() throws Exception{
          ..............................
    ④ return myServer.accept();
   }
....
```

execute() 메소드에서는 dataChannel을 생성(①)하고, dataChannel을 연결할 socket을 openSocket() 메소드를 호출함으로써 생성(②,④)한다. 생성된 socket과 dataChannel을 통하여 데이터를 전송한다(③).

3. 병렬 전송의 성능

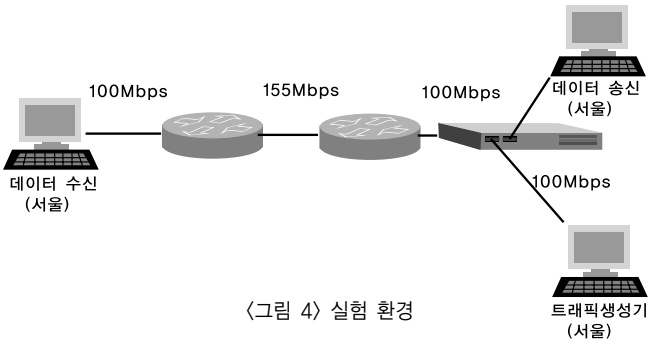
병렬 스트림을 이용하여 데이터를 전송할 경우 성능의 향상 정도를 알아보기 위해, GridFTP 분석 내용을 토대로 간단한 병렬 전송 툴을 구현하여 실험하였다. 500MB의 데이터를 단일 스트림과 다중 스트림(2~32개)을 이용하여 전송하였고, 각각의 경우에 송신측의 전송 시간을 측정하였다. 네트워크의 부하에 따라, 그리고 성능이 우수한 호스트(Pentium IV)에서 데이터를 전송할 경우, 낮은 성능의 호스트(Pentium III)를 이용하여 데이터를 전송할 경우에 전송 시간을 측정하였다.

(1) 실험 환경

KREONET 155M 백본 망과 100Mbps fast ethernet 사용하는 네트워크 망에 3대의 컴퓨터를 연결하고 이를 이용하여 여러 실험을 구성하였다.
네트워크 환경은 〈그림 4〉와 같이 수원에 한대의 호스트와 서울에 2대의 호스트로 구성되어 있으며, 수원에 위치해 있는 호스트의 성능은 Pentium III 800MHz 이며, 서울에 위치해 있는 두 호스트의 성능은 Pentium III 733MHz 이다.

또한, 송신측 호스트의 성능에 따라 전송 시간을 변화를 알아보기 위해 Pentium IV 1.53GHz 성능의 호스트를 실험 중간에 대체 사용하였다.

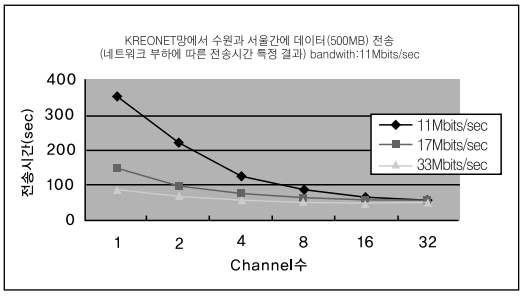
서울에 위치한 호스트 중 한대는 트래픽을 발생시키는데 사용되었고, 수원에 위치한 호스트와 서울에 위치한 나머지 한 대의 호스트 사이에는 실제 스트림을 생성하고 데이터를 전송하였다.



〈그림 4〉 실험 환경

(2) 병렬 전송에 의한 데이터 전송 능력의 변화

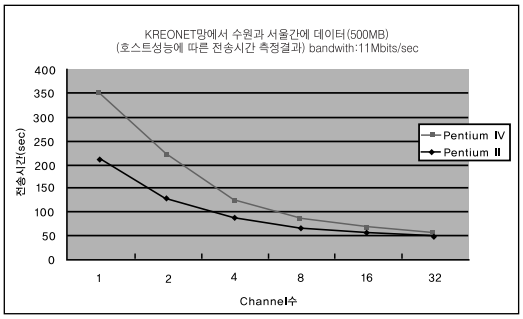
〈그림 5〉는 500MB의 데이터를 전송하고, 네트워크의 부하에 따라, 스트림의 개수에 따라 측정된 전송 시간을 그래프로 나타내고 있다. 스트림의 개수가 32개에 가까워질수록 전송 시간이 단축되고 있음을 알 수 있고, 가용 대역폭이 11Mbits/sec인 경우(네트워크 부하가 많은 경우) 전송 시간이 크게 단축되고 있음을 알 수 있다.
대역폭이 100Mbits/sec인 네트워크 망에서 대역폭을 100% 활용한다고 가정하면 500MB의 데이터를 전송하는데 약 40초정도 소요된다. 〈그림 5〉의 그래프에서 32개의 스트림으로 데이터를 전송할 경우를 보면 전송 시간이 약 50초로 대역폭을 대부분 이용하고 있음을 알 수 있다.



〈그림 5〉 병렬 전송시 데이터 전송 시간 측정 결과

또한, 가용 대역폭이 커질수록 전송 시간이 단축되는 폭이 작아지는 현상을 볼 수 있다. 이는 병렬 전송이 네트워크에 부하가 많을수록 큰 효과를 얻을 수 있음을 의미한다.
호스트의 성능이 병렬 전송에 미치는 영향을 알아보기 위해 성능이 다른 두 호스트에서 같은 크기의 데이터를 각각 전송하고 전송 시간을 측정하였다. 그 결과를 〈그림 6〉에서 볼 수 있다

단일 스트림을 이용하여 데이터를 전송할 경우에는 약 150초 정도의 전송 시간의 차이를 보이다가 스트림의 수가 32에 가까워질수록 차이가 적어지는 것을 볼 수 있다. 이는 병렬 스트림을 이용할 경우, 호스트의 성능이 전송 능력에 크게 영향을 미치지 못함을 나타내낸다.

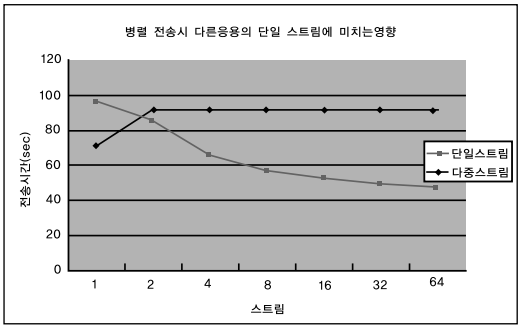


〈그림 6〉 호스트 성능에 따른 병렬 전송 성능 비교

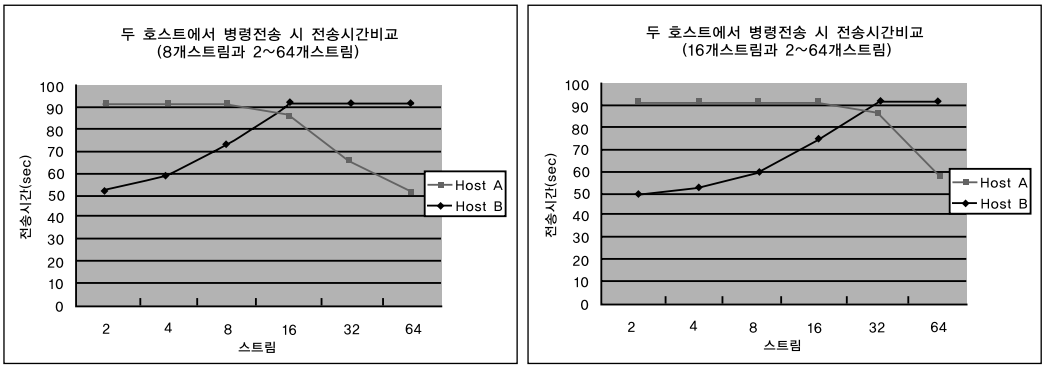
4. 병렬 전송의 문제점

〈그림 7〉은 병렬 전송이 다른 응용프로그램의 데이터 전송에 미치는 영향을 알아보기 위해 실험한 결과를 나타내고 있다. 단일 스트림은 100MB의 데이터를 전송하고, 다중 스트림은 500MB의 데이터를 1개~64개의 스트림을 이용하여 전송하면서, 각각의 전송 시간을 측정하였다.
단일 스트림을 이용하여 데이터를 전송할 경우 약 70초 정도의 시간이 소요되던 것이, 한 응용 프로그램에서 병렬(2~64개 스트림 이용)로 데이터를 전송하게 되면, 전송 시간이 약 90초로, 20초 정도의 지연이 발생되었다. 결국 64개의 스트림을 이용하여 데이터를 전송할 경우 500MB의 데이터가 100MB의 데이터보다 약 40초 빨리 도착하는 현상이 나타났다.
〈그림 8〉은 두 응용 프로그램이 데이터를 병렬로 전송할 경우에 병렬 전송 간에 미치는 영향을 알아보기 위해 실험한 결과이다. 〈그림 8〉의 (a)에서 Host A는 8개의 스트림을 이용하여 500MB 데이터를 전송하고 측정된 시간을 나타낸 것이고,

(b)의 Host A는 16개의 스트림을 이용하여 데이터를 전송하고 측정된 시간을 나타낸 것이다. 또한, (a)와 (b)에서 Host B는 스트림을 2개에서 64개까지 늘여가면서 측정한 전송 시간을 나타낸 것이다.
(a)를 보면, 8개의 스트림을 이용하여 데이터를 전송할 경우 약 50초의 전송 시간을 기록하다가 다른 응용의 병렬 스트림 개수가 늘어나면서 전송 시간 지연이 일어나는 현상을 볼 수 있다. 다른 응용의 스트림이 16개에 이르게 되면 전송 시간이 엇갈리면서 다른 응용에서 전송하는 데이터가 더 빠른 시간 내에 목적지에 도달하게 된다. 그림 (b)에서도 마찬가지로 32개의 스트림에서 전송 시간이



〈그림 7〉 병렬 전송이 단일 스트림을 이용한 전송에 미치는 영향



(a) 8개 스트림과 비교 (b) 16개 스트림과 비교

〈그림 8〉 병렬 전송간의 간섭 현상

엇갈리는 현상이 나타나면서 다른 응용의 전송 시간이 단축되는 현상이 나타난다.
위의 실험 결과들은 하나의 스트림이 전체 대역폭의 1/n 만큼을 사용함으로써 발생되는 현상으로 분석할 수 있다. 따라서 스트림의 개수가 많을수록 많은 대역폭을 차지하게 되고, 이로 인해 다른 응용프로그램들의 데이터 전송 지연, 혹은 손실이라는 심각한 문제를 초래할 수 있다.

5. 결론 및 향후 연구 과제

데이터 그리드는 전 세계 통합 데이터 관리를 목표로 설계되어졌다. 이러한 데이터 그리드의 핵심 기술 중에 하나인 GridFTP는 기존의 FTP에서 확장되어 정의, 구현되어졌으므로 TCP를 기반으로 데이터 전송이 이루어진다. 따라서 GridFTP는 TCP의 문제점을 그대로 가지고 있어, GridFTP에서는 이를 해결하기 위해 병렬 전송과 스트라이핑 기술을 지원하고 있다.
본 글에서는 여러 실험을 통하여 병렬 전송 기술의 효과를 증명하였으며, 병렬 전송으로 인해 발생하는 간섭 현상에 대한 문제를 제기하였다. 이러한 간섭 현상으로 인해 실시간으로 전송되어야 할 데이터, 예를 들어, 동영상 등의 멀티미디어와 같은 데이터가 전송 지연되는 문제가 발생될 수 있다.
따라서 병렬로 데이터를 전송하기 위해서는 간섭을 최소로 줄이면서, 효율을 최대로 늘릴 수 있는 최적의 스트림 수를 결정하는 메커니즘 연구와 QoS 지원에 대한 연구가 절실히 필요한 실정이다.